EXHIBIT 1

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.,

Plaintiff,

v.

GOOGLE INC.,

Defendant.

Case No. CV 10-03561 WHA

EXPERT REPORT OF ROBERT ZEIDMAN

f. Android Ice Cream Sandwich (API Level 14) contains 11,692 declaring code statements from Java SE 5;

g. Android Jelly Bean (API Level 15) contains 11,692 declaring code statements from Java SE 5;

h. Android Jelly Bean (API Level 16) contains 11,693 declaring code statements from Java SE 5;

i. Android Jelly Bean (API Level 17) contains 11,694 declaring code statements from Java SE 5;

j. Android Jelly Bean (API Level 18) contains 11,720 declaring code statements from Java SE 5;

k. Android Kit Kat (API Level 19) contains 11,730 declaring code statements from Java SE 5;

l. Android Kit Kat (API Level 20) contains 11,731 declaring code statements from Java SE 5;

m. Android API Lollipop (Level 21) contains 11,734 declaring code statements from Java SE 5;

n. Android Lollipop (API Level 22) contains 11,735 declaring code statements from Java SE 5;

o. Android Marshmallow (API Level 23) contains 11,717 declaring code statements from Java SE 5;

44. I further conclude that Google copied the SSO for the 37 Java API Packages in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

45. I also conclude that Google copied 816 lines of declaring code from Java SE 6 and 223 lines of declaring code from Java SE 7 in Android Gingerbread (API Level 9) through

Android Marshmallow (API Level 23).

46. Finally, I conclude that by enabling Android apps to be run on Chrome OS devices such as Chromebooks, Google has copied the declaring code and SSO of the 37 Java API Packages.

VI. ANALYSIS

A. Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22)

47. The following sections describe the tools and procedures I used to perform an analysis to identify instances where Google copied the declaring code and SSO from the 37 Java API Packages in Android. I performed the analysis described below on two versions of Android: (1) Android Gingerbread (API Level 9) and Android Lollipop (API Level 22). I also performed another analysis, as described in Section VI.B, to confirm that Google also copied declaring code and SSO in other versions of Android.

48. Note that the source code files for the 37 Java API Packages at issue are organized in the same directories and files in Android as they are in Java. For example, package `java.lang.reflect` was found in the Java directory path `\jdk1.5.0\src\java\lang\reflect\` and in the Android directory path `\libcore\luni\src\main\java\java\lang\reflect` where the last part of the directory paths was identical. These last parts of the directory paths represent the package names.

1. *Declaring Code*

49. The source code analysis focused on “declaring code” in the Java and Android platforms. “Declaring code” refers to source code that introduces or specifies an entity in the Java Platform. This includes declarations for packages, imports, classes, fields, methods, constructors, interfaces, and annotations. A summary of the various types of declaring code and

the list of copied declaring code in the final formatted spreadsheet. Also, on the occasion I found declaring code that was in fact copied but required a manual copy and paste of source code into the spreadsheet, I pasted in the clarifying declaring source code and noted the declaring code pair as copied in the final formatted spreadsheet. I then formatted the spreadsheet and sorted it based first on package name, then filename, and lastly the first line number of the declaring code in the Java source code.

69. I ran the process described in step 1 on the source code for Java SE 5, Android Gingerbread (API Level 9), and Android Lollipop (API Level 22). A spreadsheet summarizing the declaring code in Android Gingerbread (API Level 9) that was copied from Java SE 5 is attached as Exhibit F. A spreadsheet summarizing the declaring code in Android Lollipop (API Level 22) that was copied from Java SE 5 is attached as Exhibit S.

70. Based on my analysis, it is my opinion that Google copied the declaring code from the 37 Java API Packages in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22) from Java SE 5.

4. *Presence of Private Java Declarations.*

71. Exhibits F and S each contain a notable amount of declaring code that is declared `private`. Such `private` declarations are not part of the Java API specification documentation that are publicly accessible and cannot be directly used by software developers using the 37 Java API Packages in their own software. In other words, the `private` declaring code acts in many ways like implementing code.

72. Because the `private` declaring code is not part of the public Java API, Google could not have copied this declaring code from Oracle's online Java API documentation.

73. I understand from Oracle's counsel that Oracle filed a copyright registrations for Java SE 5 (and likewise for SE 6 and SE 7) and that this is *prima facie* evidence of Oracle's ownership of the private declarations.

74. I also independently attempted to verify ownership of these private declarations. First, I looked at the copyright notice in the comments of the source code files in Java SE 5 whose `private` declarations Google copied in Android. These notices listed Sun or Oracle as the owner, suggesting that the declaring code belongs to Oracle.

75. Second, I attempted to eliminate the possibility that there was a common third-party source from which Java SE 5 and Android obtained the privately declared source code (in particular, code from the Apache Harmony project, where Google got much of the source code for the 37 Java API Packages). Harmony was not an Apache project until 2005, so the Java SE 5 files predate the project entirely.[3] I examined the metadata and copyright and license notices for the relevant source code files in Java SE 5 and the files from Apache Harmony. The Java SE 5 files are dated September 2004, whereas the Harmony files are dated October 2007. *See* http://archive.apache.org/dist/harmony/milestones/5.0/M3/apache-harmony-src-r580985-snapshot.zip. Since Sun/Oracle created its files three years before Apache did, the Apache Harmony project could not be a common third party source for this code. Based on the above, I believe the logical conclusion is that Google copied this code from Sun rather than a common source third-party source.

B. Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Other Versions of Android

76. The analysis I performed to identify the declaring code that Google copied from the 37 Java API Packages from Java SE 5 into Android, described in Section VI.A.2 above, covered two versions of Android: Android Gingerbread (API Level 9) and Android Lollipop (API Level 22).

77. It makes sense that the same declaring code should be in other versions of Android that came between Android Gingerbread (API Level 9) and Android Lollipop (API Level 22),

[3] http://harmony.apache.org/newshistory.html (accessed Jan. 7, 2016); http://wiki.apache.org/harmony/ (accessed Jan. 7, 2016).

code that Google copied from Java SE 5 into Android. They did not. The changes to the 37 Java API Packages for these versions of Android are attached as Exhibits Z (API Level 11), AA (API Level 14), BB (API Level 19), and CC (API Level 21).

105. The changes to the 37 Java API Packages in Exhibits Y through CC included additions (*e.g.* new classes or methods) and changes (*e.g.* changing a method from being abstract to non-abstract), and removals (e.g. removing a method from a class).

106. Where there were *additions* to the packages, these changes are not relevant to whether the declaring code in Android was copied from Java SE 5, though it may be relevant to copying from later versions of Java. *See* Section VI.D, below.

107. Where there were *changes* to the packages, I looked to see if the specific change suggested that Google did not copy the declaring code. None of these changes alters my opinion that Google copied declaring code the 37 Java API Packages from Java SE 5 in Android.

3. *Google Copied Declaring Code from Java SE 5 in Android Honeycomb.*

108. As described in Paragraph 30, above, I was unable to analyze the Android Honeycomb (API Levels 11 through 13) source code, however I am able to identify the declaring code Google copied using the results from my analysis of Android Gingerbread (API Level 10) and Google's Diff Reports.

109. Based on my earlier analysis, I identified 11,723 lines of declaring code in Android Gingerbread (API Level 10) that were copied from Java SE 5. *See* Exhibit G.

110. The Diff Report for Android Honeycomb (API Level 11) identified changes to or additions of 29 lines of declaring code in `java.lang` and `java.util`. I confirmed that none of these changes were present in Java SE 5. On this basis, it is my opinion that there are 11,723 lines of declaring code in Android Honeycomb (API Level 11) that were copied from Java SE 5. *See* Exhibit H.

111. In the Diff Reports for Android Honeycomb (API Level 12) and Android Honeycomb (API Level 13), Google claims that it did not make any changes to the 37 Java API Packages. On

associated with a package.

117. I visually inspected the Java and Android documentation for each package in the 37 Java API Packages. I observed that the classes, interfaces, exceptions, enums, errors, and annotation types are organized in the same hierarchical manner, within packages of the same name, in Android as they occur in Java. Exhibit U shows a side-by-side comparison showing this identical hierarchical organizational structure for each of the 37 Java API Packages.

118. My visual comparison of the SSO of Java SE 5, which is expressed through this online documentation, shows the same hierarchical organization for the classes.

119. From the analysis above, I conclude that Google's copying of the declaring code for the 37 Java API Packages into Android also led to copying of the SSO for Java SE 5.

D. Google Has Continued to Copy New Declaring Code and SSO of the 37 Java API Packages from Java SE 6 and Java SE 7

120. In addition to copying declaring code for the 37 Java API Packages from Java SE 5, Google has also copied declaring code from later versions of Java: Java Platform, Standard Edition 6 ("Java SE 6") and Java Platform, Standard Edition 7 ("Java SE 7").

121. I understand Java SE 6 was released in December 2006 and Java SE 7 was released in July 2011. Thus at least some of the copying of the declaring code from Java SE 6 and Java SE 7 occurred after Oracle filed this lawsuit in August 2010.

122. I arrived at this conclusion by further examining the results of my analysis of the Android Diff Reports as described in Paragraphs 96–107. For each change to the 37 Java API Packages in Android, I looked to see when that change appeared in a version of Java. I did this by looking at the online API documentation for Java.[6]

123. For example, in the Diff Report for Android Kit Kat (API Level 19), it identifies class

[6] In particular, I looked at documentation for Java SE 1.3.1 (https://docs.oracle.com/javase/1.3/docs/api/, download required), Java SE 1.4.2 (https://docs.oracle.com/javase/1.4.2/docs/api/, download required), Java SE 5 (https://docs.oracle.com/javase/1.5.0/docs/api/), Java SE 6 (http://docs.oracle.com/javase/6/docs/api/), Java SE 7 (http://docs.oracle.com/javase/7/docs/api/), and Java SE 8 (http://docs.oracle.com/javase/8/docs/api/).

`GCMParameterSpec` being added to package `javax.crypto.spec`. The online documentation for Java SE 7 identifies this class as being added in Java SE 7 and it is not present in the Java SE 6 online documentation. Therefore, I can conclude that this class (and the related constructors and methods) were introduced in Java SE 7.

124. Based on my analysis, I identified 816 instances of Google copying declaring code introduced in Java SE 6 into Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23). A summary of my findings for Java SE 6 is attached as Exhibit V. I also identified 223 instances of Google copying declaring code introduced in Java SE 7 into Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23). A summary of my findings for Java SE 7 is attached as Exhibit W.

125. By copying the declaring code from Java SE 6 and Java SE 7 in these versions of Android, Google also copied the intricate relationships of the classes, methods, interfaces, and fields in those packages, thus copying the SSO of Java SE 6 and Java SE 7.

E. Google Copied the Declaring Code and SSO of the 37 Java API Packages in Google's "ARC" Product

1. *The App Runtime for Chrome ("ARC") Technology Allows Android Apps to Run on Chrome OS Devices*

126. The App Runtime for Chrome (ARC) is a runtime environment created by Google that allows the Android runtime to be used on Chrome OS devices.[7] One of the primary uses for this feature is the ability to run Android apps on the Chrome OS platform.

127. In order to use ARC to run Android apps, an app called ARC Welder is used. ARC Welder allows APK files[8] to be repacked such that they can be used with the ARC technology.[9]

---

[7] See https://developer.chrome.com/apps/getstarted_arc for a brief overview of ARC.

[8] APK files contain all parts of an Android app in a single file, and can be run directly on the Android operating system.

Once ARC Welder has repackaged the APK file, the Android app can be run using a Chrome OS device with ARC installed. Note that ARC refers to the actual runtime, while ARC Welder is simply the repackaging tool.

128. It is important to note that ARC Welder does not actually convert the resources from the Android app from one format into another or transform the app in any way, it simply reorganizes the app's existing resources such that they can be accessed and executed by Chrome OS.

129. Along with ARC, Google also released a set of four apps to the Chrome Web Store that were converted from Android apps in the Google Play Store. These four apps are Duolingo, Evernote, Sight Words, and Vine.[10] [11] These apps, along with other Android apps, are repackaged and available for download and use on Chrome, and are referred to on the Chrome Web Store as "ARC apps."

130. The ARC technology is separate from the ARC Welder tool. When the ARC Welder tool is installed onto a Chrome OS device, the installation process checks to see whether ARC is installed, and automatically installs ARC if it is not. The same ARC installation check takes place when downloading any ARC app from the Chrome Web Store.

2. *The ARC Runtime Introduces Android Runtime Components into Chrome OS Devices*

131. Since a Chrome OS device with ARC installed is able to run Android apps, this implies that there must be certain runtime software components present in a Chrome OS device

---

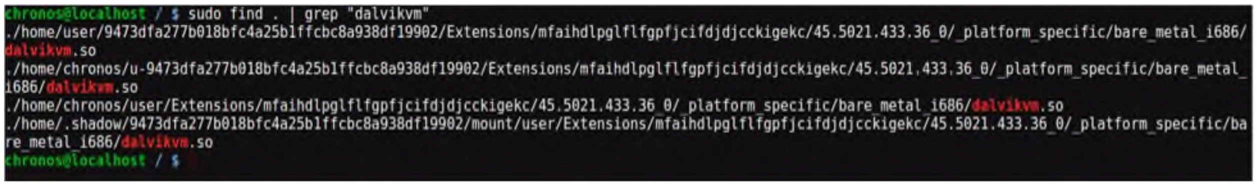
[9] See https://chrome.google.com/webstore/detail/arc-welder/emfinbmielocnlhgmfkkmkngdoccbadn for the download page for the ARC Welder app on the Google Play Store.

[10] See http://chrome.blogspot.com/2014/09/first-set-of-android-apps-coming-to.html for the Chrome Blogspot posting announcing the release of the first four Android apps packaged for use on Chrome and released to the Chrome Web Store.

[11] See https://chrome.google.com/webstore/detail/duolingo/ebnhfamfopiobpaehmebmfjcgkaogihe?hl=en-US (Duolingo), https://chrome.google.com/webstore/detail/evernote/dhfolfjkgpeaojbiicgheljefkfbbfkc?hl=en-US (Evernote), https://chrome.google.com/webstore/detail/sight-words/ikmpccnfemdkmmoejgmdajnkbidifpgh?hl=en (Sight Words), and https://chrome.google.com/webstore/detail/vine/plfjlfohfjjpmmifkbcmalnmcebkklkh?hl=en-US (Vine) for the pages on the Chrome Web Store from which these ARC apps can be downloaded.

with ARC that are the same as those used by the Android operating system.

132. Searching through the file system of an ARC-enabled Chromebook running the latest version of Chrome OS[12] revealed that binary files with names referencing the Dalvik virtual machine and core Java classes are present in the Chromebook file system. These files are `dalvikvm.so`, `libjavacore.so`, and `libjavacrypto.so`. See Figure 1 for the results of a search through the file system, which returns results showing binary files that appear to be associated with the Dalvik virtual machine.



```
chronos@localhost / $ sudo find . | grep "dalvikvm"
./home/user/9473dfa277b018bfc4a25b1ffcbc8a938df19902/Extensions/mfaihdlpglflfgpfjcifdjdjcckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/
dalvikvm.so
./home/chronos/u-9473dfa277b018bfc4a25b1ffcbc8a938df19902/Extensions/mfaihdlpglflfgpfjcifdjdjcckigekc/45.5021.433.36_0/_platform_specific/bare_metal_
i686/dalvikvm.so
./home/chronos/user/Extensions/mfaihdlpglflfgpfjcifdjdjcckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/dalvikvm.so
./home/.shadow/9473dfa277b018bfc4a25b1ffcbc8a938df19902/mount/user/Extensions/mfaihdlpglflfgpfjcifdjdjcckigekc/45.5021.433.36_0/_platform_specific/ba
re_metal_i686/dalvikvm.so
chronos@localhost / $
```

Figure 1: Shared object files from the Dalvik Virtual Machine found in the file system of a Chromebook device indicate the presence of Android runtime components.

133. The contents of these .so files are low-level object code that is not human-readable by direct inspection. In order to make sense of the contents of these binaries, the readelf command was used in order to interpret contents of these binary files.[13]

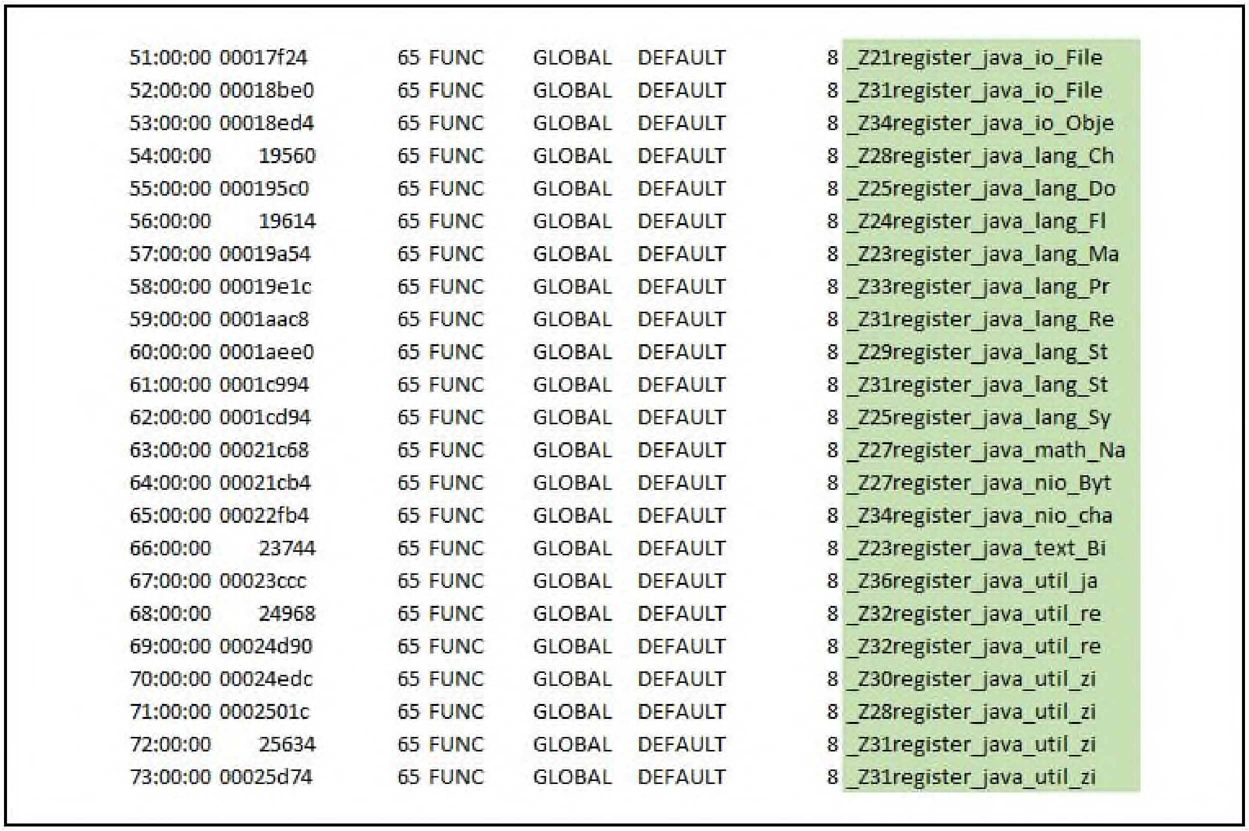
134. A selected portion of the output of the `readelf` command for the libjavacore.so file is shown in Figure 2. The full `readelf` output for this binary file is presented in Exhibit X. Each row of the output provides detailed information about a single symbol in the ELF file. The rightmost part of the entry gives the name of the symbol itself. From the information in the figure, it is clearly evident that the names of these symbols are virtually identical to the names of some of the infringed packages (`java.io`, `java.lang`, `java.nio`, `java.text`, and

[12] This test used an Acer Chromebook 11, running in developer mode and using the 11 December 2015 build of Chrome OS. See https://store.google.com/product/acer_chromebook_11 for the product description page of this model on the Google Store.

[13] The `readelf` command is a GNU binary utility used to display information about ELF (Executable and Linkable Format) object files, a common file format for object code and shared libraries. See https://sourceware.org/binutils/docs/binutils/readelf.html for a detailed description of the `readelf` command. Since ELF-type files (such as `.so` files) are not human-readable, the `readelf` command provides a means by which the contents of these types of files can be displayed in a human-readable manner. In this case, the `readelf -all` command was used, which lists the most complete information about the contents of the ELF files.

`java.util`).

135. This shows that the file contains compiled binary versions of classes derived from the 37 Java API Packages at issue. Because such binary files are present in the file system of the Chromebook following installation of ARC technology, the ARC runtime environment makes use of the infringing material in order to allow Android applications to be run on Chrome OS devices.



```
51:00:00 00017f24    65 FUNC   GLOBAL  DEFAULT       8 _Z21register_java_io_File
52:00:00 00018be0    65 FUNC   GLOBAL  DEFAULT       8 _Z31register_java_io_File
53:00:00 00018ed4    65 FUNC   GLOBAL  DEFAULT       8 _Z34register_java_io_Obje
54:00:00       19560 65 FUNC   GLOBAL  DEFAULT       8 _Z28register_java_lang_Ch
55:00:00 000195c0    65 FUNC   GLOBAL  DEFAULT       8 _Z25register_java_lang_Do
56:00:00       19614 65 FUNC   GLOBAL  DEFAULT       8 _Z24register_java_lang_Fl
57:00:00 00019a54    65 FUNC   GLOBAL  DEFAULT       8 _Z23register_java_lang_Ma
58:00:00 00019e1c    65 FUNC   GLOBAL  DEFAULT       8 _Z33register_java_lang_Pr
59:00:00 0001aac8    65 FUNC   GLOBAL  DEFAULT       8 _Z31register_java_lang_Re
60:00:00 0001aee0    65 FUNC   GLOBAL  DEFAULT       8 _Z29register_java_lang_St
61:00:00 0001c994    65 FUNC   GLOBAL  DEFAULT       8 _Z31register_java_lang_St
62:00:00 0001cd94    65 FUNC   GLOBAL  DEFAULT       8 _Z25register_java_lang_Sy
63:00:00 00021c68    65 FUNC   GLOBAL  DEFAULT       8 _Z27register_java_math_Na
64:00:00 00021cb4    65 FUNC   GLOBAL  DEFAULT       8 _Z27register_java_nio_Byt
65:00:00 00022fb4    65 FUNC   GLOBAL  DEFAULT       8 _Z34register_java_nio_cha
66:00:00       23744 65 FUNC   GLOBAL  DEFAULT       8 _Z23register_java_text_Bi
67:00:00 00023ccc    65 FUNC   GLOBAL  DEFAULT       8 _Z36register_java_util_ja
68:00:00       24968 65 FUNC   GLOBAL  DEFAULT       8 _Z32register_java_util_re
69:00:00 00024d90    65 FUNC   GLOBAL  DEFAULT       8 _Z32register_java_util_re
70:00:00 00024edc    65 FUNC   GLOBAL  DEFAULT       8 _Z30register_java_util_zi
71:00:00 0002501c    65 FUNC   GLOBAL  DEFAULT       8 _Z28register_java_util_zi
72:00:00       25634 65 FUNC   GLOBAL  DEFAULT       8 _Z31register_java_util_zi
73:00:00 00025d74    65 FUNC   GLOBAL  DEFAULT       8 _Z31register_java_util_zi
```

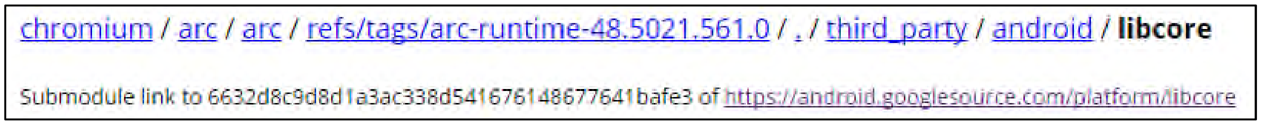
Figure 2: Certain symbols from the `readelf` output of binary file libjavacore.so contain compiled resources from the 37 infringed packages.

3. *The ARC Runtime Build Process Contains the 37 Java API Packages.*

136. The publicly available code repository for ARC Runtime contains a folder called `third_party/android/libcore` folder[14] that links directly to the called

[14] The libcore folder for ARC Runtime can be accessed at https://chromium.googlesource.com/arc/arc/+/refs/tags/arc-runtime-48.5021.561.0/third_party/android/

`platform/libcore` folder in in Android Open Source Project Repository[15], as shown in Figure 3, which contains the Android source code for the 37 Java API Issues.


chromium / arc / arc / refs/tags/arc-runtime-48.5021.561.0 / . / third_party / android / **libcore**

Submodule link to 6632d8c9d8d1a3ac338d5416761486776 41bafe3 of https://android.googlesource.com/platform/libcore

Figure 3. Libcore folder in the publicly available code repository for ARC runtime

137. Furthermore, the build scripts for the ARC Runtime references, also found in the publicly available code repository, references the code that contains the 37 Java API Packages in the `third_party/android/libcore` folder.

4. *Android Apps Made Available for Use on Chrome through ARC Rely on and Use the 37 Java API Packages*

138. As was mentioned before, Evernote was one of the four original ARC apps released to the Chrome Web Store as Android apps repackaged and ready for use on Chrome OS. Evernote is one of the most frequently downloaded Android apps available on the Google Play Store, having been downloaded to hundreds of millions of Android devices to date.[16]

139. The Evernote app architecture implements material from the 37 Java API packages. The advent of ARC and ARC apps makes the Evernote app, as well as its leveraging of infringed content, available for use on Chrome OS devices. To this end, an analysis was conducted to determine which of the 37 Java API packages are used in Evernote.

140. To perform this analysis, the Evernote Android APK file was downloaded from the Google Play Store using the Firefox APK Downloader plugin.[17] Following this, a series of reverse-engineering tools were used to convert the apk file back into source code such that its

[15] The libcore folder in AOSP can be accessed at https://android.googlesource.com/platform/libcore

[16] This was determined by retrieving a dataset showing user rating and download metrics for all apps available on the Google Play Store. Based on this data, the Evernote app reports having somewhere between 100 million and 500 million app downloads as of 21 December 2015.

[17] See https://play.google.com/store/apps/details?id=com.evernote&hl=en for the Google Play Store page for the Evernote app.

underlying dependencies could be extracted.[18] A tool called Understand was then used to extract the dependency structure of the app from the reverse-engineered source code.[19]

141. The results of the analysis show the Evernote app using 13 of the 37 Java API packages in its source code: `java.net`, `java.nio`, `java.lang.reflect`, `javax.net`, `java.lang.ref`, `java.io`, `java.security`, `java.text`, `java.util.zip`, `java.util.regex`, `java.lang`, `java.util`, and `javax.crypto`.

142. This shows that ARC makes available to Chrome OS devices an app that has been downloaded hundreds of millions of times, and leverages the functionality of 13 of the 37 Java API packages through use of the Android runtime.

143. In a broader sense, ARC expands the extent to which the Android platform benefits from its appropriation of the 37 infringed packages. It does so both by actually implementing the infringed content in the ARC technology itself, as well as by providing a medium of access by which Chrome OS devices can use app content that leverages this infringed material as well.

VII. CONCLUSION

144. I have analyzed the source code for both Java SE 5 and Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23), looking for instances where declaring code for the 37 Java API Packages from Java SE 5 appears in Android. Based on this analysis, I conclude the following:

a. Android Gingerbread (API Level 9) contains 11,723 declaring code statements from Java SE 5;

b. Android Gingerbread (API Level 10) contains 11,723 declaring code statements from Java SE 5;

[18] This process begins with simply unpacking the APK file to extract its contents, and using a tool called dex2jar (https://github.com/pxb1988/dex2jar) to convert the .dex bytecode contents of the APK file back into .class bytecode format. Note that this does not transform the contents in any functional manner, but simply converts from one format to another. After this, the output of the dex2jar tool is processed by a tool called CFR (http://www.benf.org/other/cfr/) which de-compiles the bytecode into its original Java source code.

[19] See https://scitools.com/ for a description of the various functions of the Understand static code analysis tool.

Java SE 5;

145. I further conclude that Google copied the SSO for the 37 Java API Packages in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

146. I also conclude that Google copied 816 lines of declaring code from Java SE 6 and 223 lines of declaring code from Java SE 7 in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

147. Finally, I conclude that by enabling Android apps to be run on Chrome OS devices such as Chromebooks, Google has copied the declaring code and SSO of the 37 Java API Packages.

148. It is my understanding that discovery in this case is ongoing. Accordingly, I reserve the right to supplement or amend my opinions in light of any additional evidence, testimony, or information that may be provided to me after the date of this report. I also reserve the right to supplement or amend my opinions in response to any expert reports served by any party in the lawsuit.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on: January 8, 2016

[illegible]

Robert Zeidman